Dokumentation Hamming-Code

# Ziel

Mittels des Hamming-Codes können Bitfehler bei der Übertragung von Daten detektiert werden. Dieser Code ist linear und kann nicht nur 1-Bitfehler erkennen, sondern diese auch lokalisieren. Falls mehrere Fehler zur gleichen Zeit auftreten, können diese nicht zuverlässig erkannt oder lokalisiert werden, da der Code von linearer Struktur ist.

Durch das Hammingmodul können in Python leicht Hamming-Codes leicht codiert und dekodiert werden. Der Hamming-Code ist ein fehlerkorrigierender Zusatz, den man mit Daten verschicken kann, um eventuell fehlerhafte Übertragungen zu vermeiden.

# Nutzung:

Als erstes kann man das Hamming-Modul importieren:

from Hamming import encode\_hamming, decode\_hamming

Von diesem Modul werden zwei Funktionen verwendet: encode\_hamming, decode\_hamming

Als Eingabe nutzen die beiden Funktionen Datentypen, die Daten abspeichern, sodass sie in leserlicher Form als Binärfolge ausgegeben und verwertet werden können, z. B. Strings oder Listen. Durch die Funktion encode\_hamming werden diese daten codiert. Anschließend können sie mit der Funktion decode\_hamming decodiert werden. Ausgegeben wird diese Decodierung in dem gleichen Datentyp der Eingabe.

Beispiel:

bits = “100101”

ham = encode\_hamming(bits)

bits = decode\_hamming(ham)

print(bits)

# Rückgabe: 100101

Von dem modul werden zwei funktionen verwendet (encode\_hamming & decode\_hamming). Sie nehmen beide ein bits-ähnliches objekt wie ein string, eine liste oder ein bitarray. Sie kreieren codes oder dekodieren diese dann. Die rückgabe ist ein bits-ähnliches objekt mit dem datentyp der Eingabe.

# Funktionsweise:

Um 1-Bitfehler bei der Übertragung eindeutig identifizieren zu können, werden Paritätsbits genutzt. Diese Bits stehen immer an der -ten Stelle. Alle anderen Stellen werden fortlaufend mit der binären Nachricht einzeln befüllt, wobei die Stellen der Paritätsbits übersprungen werden. Alle Stellen, die mit einer binären 1 gefüllt werden, werden binär umgewandelt und mit dem logischem Operator XOR kombiniert. Die Bits dieser entstandenen Bitsequenz werden fortlaufend auf die Stellen der Paritätsbits verteilt. Nun kann dieser Hamming-Code verschickt werden.

Bei der Decodierung werden die Binärzahlen für die Paritätsbits auf gleiche Weise wie bei der Codierung berechnet. Allerdings werden hierbei auch die Paritätsbits, die eine 1darstellen, mit den restlichen Stellen, die eine 1 enthalten, durch XOR kombiniert. Liegt am Ende eine 0 vor, so war die Übertragung fehlerfrei. Ist allerdings ein 1-Bitfehler aufgetreten, so entspricht die Stelle des Fehlers des Ergebnisses der XOR-Kombination. Liegen allerdings mehrere Fehler vor, so können die Positionen nicht eindeutig bestimmt werden.

Um Hamming-Codes zu erstellen, fügt man zu den Bits an den Stellen eine Zahl ein. Die Zahl, die in dieser Verteilung eingefügt wird, wird generiert, indem die Zahlen der Stellen aller Einsen in den Bits (Stellen nach dem hypothetischem Einfügen des Codes) mit einem XOR kombiniert werden.

Für die Dekodierung berechnet man diese Zahl wieder mit den Datenbits. Diese und die Korrekturbits können dann mit einem XOR kombiniert werden. Kommt eine 0 raus, liegen keine Fehler vor. Kommt etwas anderes raus, gibt es entweder einen Fehler, in welchem Fall die Ausgabe dieser Rechner die Position des falschen Bits ist, oder mehrere Fehler, in welchem Fall man die k orrekte Ausgabe nicht bestimmen kann.

# Beispielprogramm

Als Beispielprogramm liegt example.py vor. Als Input kann ein eingegeben werden, um zu codieren, für decodieren und , um das Programm zu stoppen. Falls oder eingegeben werden, müssen anschließend die zu verarbeitenden Bits angegeben werden. Je nachdem welcher Buchstabe vorher eingegeben wurde, werden diese Bits nun codiert bzw. decodiert.

Dieses nimmt als Input für codieren, für dekodieren und , um es zu stoppen. Es fragt nach den Bits an denen gearbeitet werden soll. Es codiert oder dekodiert diese dann und gibt das Ergebnis zurück.